Density Independent Algorithms for Sparsifying k-Step Random Walks*

Gorav Jindal[1], Pavel Kolev[2], Richard Peng[3], and Saurabh Sawlani[4]

1 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
gjindal@mpi-inf.mpg.de
2 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
pkolev@mpi-inf.mpg.de
3 **Georgia Institute of Technology, Atlanta, GA, USA**
rpeng@gatech.edu
4 **Georgia Institute of Technology, Atlanta, GA, USA**
sawlani@gatech.edu

Abstract

We give faster algorithms for producing sparse approximations of the transition matrices of k-step random walks on undirected and weighted graphs. These transition matrices also form graphs, and arise as intermediate objects in a variety of graph algorithms. Our improvements are based on a better understanding of processes that sample such walks, as well as tighter bounds on key weights underlying these sampling processes. On a graph with n vertices and m edges, our algorithm produces a graph with about $n \log n$ edges that approximates the k-step random walk graph in about $m + k^2 n \log^4 n$ time. In order to obtain this runtime bound, we also revisit "density independent" algorithms for sparsifying graphs whose runtime overhead is expressed only in terms of the number of vertices.

1998 ACM Subject Classification F.2.2. [Nonnumerical Algorithms and Problems] Computations on Discrete Structures, G.3 [Probability and Statistics] Probabilistic algorithms (including Monte Carlo)

Keywords and phrases random walks, graph sparsification, spectral graph theory, effective resistances

Digital Object Identifier 10.4230/LIPIcs.APPROX/RANDOM.2017.14

1 Introduction

Random walks are some of the most natural mathematical objects, and have historically been used to model processes in fields ranging from psychology to economics. Problems related to random walks on graphs, such as shortest path and minimum cut are well studied in both static [34] and dynamic settings [21, 17]. While some of these problems, such as shortest path, aim to find a single walk, other problems such as finding flows/cuts [16] or triangle densities [4, 38] aim to capture information related to collections of walks. Algorithms and data structures for such problems often need to store, or can be sped up by, intermediate structures that capture the global properties of multi-step walks [31, 18, 1, 3]. However, many intermediate structures are inherently dense and therefore expensive to compute explicitly.

* Pavel Kolev is funded by the Cluster of Excellence "Multimodal Computing and Interaction" within the Excellence Initiative of the German Federal Government. Richard Peng and Saurabh Sawlani are partially supported by the NSF under Grant No. 1637566.

Graph sparsification is a technique for efficiently approximating a dense graph by a sparser one, while preserving some key properties such as sizes of graph cuts, distances between vertices, or linear operator properties of matrices associated with the graphs. Spectral sparsifiers are the ones which guarantee linear operator approximations, but they also inherently approximate all graph cuts. Moreover, they have various applications in graph algorithms, such as sampling from graphical models [7], solving linear systems [32, 27], sampling random spanning trees [13, 14][1] and maintaining approximate minimum cuts in dynamically changing graphs [1]. In these applications, the optimal performance is achieved by producing a sparsifier of a denser intermediate object directly, instead of generating the exact larger object. Of these intermediate objects, some of the more commonly studied are random walk matrices [32, 7]. These matrices contain the pairwise transition probabilities between vertices under k-step walks. Moreover, such matrices are dense even for sparse original graphs with k as small as 2: for instance, the 2-step walk on the n-vertex star contains an $n-1$ sized clique.

Cheng et al. [7] studied random walk sparsification, and gave a routine that produces an ϵ-spectral sparsifier (which we will formally define in Subsection 2.2) with $O(\epsilon^{-2}n\log n)$ edges for a k-step walk matrix in $O(\epsilon^{-2}k^2m\log^{O(1)}n)$ time. Our main result, which we show in Section 3, is a direct improvement of that routine:

▶ **Theorem 1** (Sparsifying Laplacian Monomials). *Given a graph G and an error $\varepsilon \in (0,1)$, there is an algorithm that outputs an ε-spectral sparsifier of G^k with at most $O(\varepsilon^{-2}n\log n)$ edges in $\widehat{O}(m+k^2\varepsilon^{-2}n\log^4 n)$ time.*[2]

We term this type of running time with most of the overhead on the number of vertices, n, as density independent. Such runtimes arise naturally in many other graph problems [15], and was first studied for graph sparsification in an earlier manuscript by a subset of the authors [22], where the authors sparsify certain Laplacian monomials (specifically, monomials where the degree is a power of 2) in $O(m\log^2 n+\epsilon^{-4}n\log^4 n\log^5 k)$ time. They also extend this to specific classes of matrix polynomials - those with coefficients induced by "mixture of discrete Binomial distributions" with similar running-times. Our algorithm can also be combined with the repeated-squaring technique in [7] to reduce the runtime dependence on k to logarithmic [8]. Additionally, if we generalize our results from monomials to general random walk polynomials [8], this would then supersede all claims from [22]. As these steps are much closer to the ideas in [7], we will focus on the small k case in this paper. Furthermore, as our sparsification algorithm has a much more direct interaction with routines that provide upper bounds of effective resistances, they can likely be combined with tools from [1] to give dynamic algorithms for maintaining G^k under insertions/deletions to G. However, as there are currently only few applications of such sparsifiers, we believe it may be more fruitful to extend the applications before further developing the tools.

Our algorithms, as with the ones from [22, 7] are based on implicit sampling of dense graphs by probabilities related to effective resistances. Our improvements rely on an a key insight from the sparse Gaussian elimination algorithm by Kyng and Sachdeva [29]: using triangle inequality between effective resistances to obtain a tighter set of probability

[1] While these manuscripts are simultaneous, the significantly earlier original proposal of density independent sparsification of walks [22], and the importance of it in the algorithm of [14] were major motivations for this paper.

[2] We use $\widehat{O}$ to denote the omission of logarithmic terms lower than the ones shown in the set. In all cases in this paper, we track terms of $\log n$ explicitly and such notation hides terms of $\log\log n$. In all these cases, this notation hides a term of at most $(\log\log n)^2$.

upper bounds. So, to sample an edge in G^k, we essentially simulate a k-step walk in G by first sampling an edge, and then "walking" along both directions to make a length k walk. A simple but crucial detail in the algorithm is the selection of that first edge. Instead of sampling it uniformly as in [7], we pick an edge e with probability proportional to the product of its weight and effective resistance (its "leverage score"). Although the change is subtle, this helps remove any sampling count dependencies on the number of edges, making a density-independent runtime possible.

Obtaining density-independent bounds is critical for graph sparsification algorithms, since they are primarily invoked on relatively dense graphs. A graph sparsification routine that produces a sparsifier with $\widehat{O}(n \log^2 n)$ edges in $\widehat{O}(m \log^2 n)$ time, such as the combinatorial algorithm given in [28], will only be invoked when $m > n \log^2 n$, which means that the running time of the algorithm is actually $\Omega(n \log^4 n)$. Additionally, a desired property of a sparsification algorithm is that applying it repeatedly does not cause a blow up in its running time. One way to achieve this is to ensure that the running time is linear in the number of edges, and the overhead is only on the number of vertices. As a result, we believe that for graph sparsification to work as a primitive for processing large graphs, a running time of $\widehat{O}(m + n \log^2 n)$, or better, is necessary.

In Section 4, we provide some steps toward this direction by outlining a better density-independent spectral sparsification algorithm. We combine ideas from previous density-independent algorithms for sparsifying graphs [26] with recent developments in tree embedding and numerical algorithms to obtain numerical sparsification routines that run in $\widehat{O}(m + n \log^4 n)$ time, and combinatorial ones that take $\widehat{O}(m + n \log^6 n)$ time. Although these routines do not involve new ideas, they utilize some of the latest machinery, and give the current best time-bounds for density-independent sparsification. Importantly, both of these routines are in turn applicable to the walk sparsification algorithm in Section 3, giving routines for sparsifying k-step walks with similar running times: the bound stated in Theorem 1 is via the numerical routine. While these results are far from what we think are the best possible, we show a variety of new algorithmic tools for designing algorithms that sparsify k-step random walks matrices.

Our methods of extending density independent sparsification to random walks play a crucial role in several other types of graph sparsification - in particular, sparsification routines requiring only an oracle that samples edges from a distribution of approximate resistances, and oracle access to approximate leverage scores. Even for the 'simpler' problem of producing cut sparsifiers of G^k, these are the only known efficient approaches.

For instance, the routines for approximately sampling and counting spanning trees from [14] rely on producing determinant preserving sparsifiers of Schur complements of graph Laplacians, which are themselves sums of random walks. Specifically, the algorithm in [14] builds a sparse graph H such that $(1-\epsilon)\det(G) \leq \det(H) \leq (1+\epsilon)\det(G)$, but requires an overhead of about $\Theta(\sqrt{n})$ samples, leading to sparsifiers with about $\Theta(\epsilon^{-2} n^{1.5})$ edges. On the other hand, the number of calls this algorithm makes to the oracles is given by $O(n^{-1} \sum_{e \in E} \ell_e)$, where ℓ_e is a value dominating the leverage score of e. Thus, to extend their algorithm to Schur complements and simultaneously guarantee that the time-bound does not blow up beyond $O(n^{1.5})$, we have to ensure that these approximate leverages scores still sum up to $O(n)$. This is similar to our requirement, and is done by picking the initial edge of the random walk with probability proportional to the product of its weight and its effective resistance, and extending it to a walk on both sides.

2 Background

We start with some background information about graphs and matrices corresponding to them. These matrices allow us to define graph approximations, as well as compute key sampling probabilities needed to produce spectral sparsifiers. Due to space constraints, we will only formally define most of the concepts. More intuition on them can be found in notes on spectral graph theory and random walks such as [12, 30].

2.1 Random Walks and Matrices

Let $G = (V, E, \boldsymbol{w})$ be a weighted undirected graph. We define its adjacency matrix $\boldsymbol{A}$ as $\boldsymbol{A}_{uv} \stackrel{\text{def}}{=} \boldsymbol{w}_{uv}$, and its degree matrix $\boldsymbol{D}$ as $\boldsymbol{D}_{uu} \stackrel{\text{def}}{=} \sum_{v \in V} w_{uv}$ and $\boldsymbol{D}_{uv} \stackrel{\text{def}}{=} 0$ when $u \neq v$. This leads to the graph Laplacian $\boldsymbol{L}_G \stackrel{\text{def}}{=} \boldsymbol{D} - \boldsymbol{A}$.

One step of a random walk can be viewed as distributing the 'probability mass' at a vertex evenly among the edges leaving it, and passing them onto its neighbors. In terms of these matrices, it is equivalent to first dividing by $\boldsymbol{D}$, and then multiplying by $\boldsymbol{A}$. Thus, the left transition matrix of the k^{th} step random walk is given by $(\boldsymbol{D}^{-1}\boldsymbol{A})^k$. The corresponding Laplacian matrix of the k-step random walk is defined by

$$\boldsymbol{L}_{G^k} \stackrel{\text{def}}{=} \boldsymbol{D} - \boldsymbol{A}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{k-1}.$$

The matrices $\boldsymbol{A}(\boldsymbol{D}^{-1}\boldsymbol{A})^{k-1}$ can be viewed as a sum over length k walks. This view is particularly useful in our algorithm, as well as the earlier walk sparsification algorithm by Cheng et al. [7] because these walks are a more 'natural' unit upon which sparsification by effective resistances is applied. Formally, we can define the weight of a length k walk $(u_0, u_1, \ldots, u_k)$ by

$$\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} \stackrel{\text{def}}{=} \frac{\prod_{i=1}^{k} \boldsymbol{w}_{u_{i-1},u_i}}{\prod_{i=1}^{k-1} \boldsymbol{d}_{u_i}}. \tag{1}$$

Straightforward checking shows that for any $u_0, u_k \in V$, the weight of the edge (u_0, u_k) in $\boldsymbol{G}^k$ is given by

$$\boldsymbol{w}^{\boldsymbol{G}^k}_{u_0,u_k} \stackrel{\text{def}}{=} \left[\boldsymbol{A}\left(\boldsymbol{D}^{-1}\boldsymbol{A}\right)^{k-1}\right]_{u_0 u_k} = \sum_{u_1,\ldots,u_{k-1}} \boldsymbol{w}_{(u_0,u_1,\ldots,u_k)}. \tag{2}$$

2.2 Spectral Approximations of Graphs

Our notion of matrix approximations will be through the $\approx$ symbol, which is in turn defined through the Löewner partial ordering of matrices. For two matrices, $\boldsymbol{A}$, and $\boldsymbol{B}$, we say that $\boldsymbol{A} \preceq \boldsymbol{B}$ if $\boldsymbol{B} - \boldsymbol{A}$ is positive semidefinite, and $\boldsymbol{A} \approx_\kappa \boldsymbol{B}$ if there exists bounds $\lambda_{\min}$ and $\lambda_{\max}$ such that $\lambda_{\min}\boldsymbol{A} \preceq \boldsymbol{B} \preceq \lambda_{\max}\boldsymbol{A}$, and $\lambda_{\max} \leq \kappa\lambda_{\min}$. This notation is identical to generalized eigenvalues, and in particular, $\boldsymbol{L}_G \approx_\kappa \boldsymbol{L}_H$ implies that all cuts on them are within a factor of κ of each other.

The adjacency matrix of a graph has several undesirable properties when it comes to operator based approximations: it can have a large number of eigenvalues at 0, which must be exactly preserved under relative error approximations. As a result, graph approximations are defined in terms of graph Laplacians. As we will discuss below, these approximations are often in terms of reducing edges. So formally, we say that a graph H is a κ-sparsifier of G if $\boldsymbol{L}_H \approx_\kappa \boldsymbol{L}_G$, and our goal is to compute an ϵ-sparsifier of the k-step random walk graph G^k.

Algorithm 1 IdealSample($G, \varepsilon, \widetilde{\boldsymbol{\tau}}$)

Input: A graph $G = (V, E, \boldsymbol{w})$, an integer k, and leverage score upper bounds $\widetilde{\boldsymbol{\tau}}_e$ that satisfy $\widetilde{\boldsymbol{\tau}}_e \geq \boldsymbol{w}_e \mathcal{R}^G_{\text{eff}}(e)$ for all edges e.
Output: An ε-sparsifier $\boldsymbol{H}$ of G with $O(\varepsilon^{-2}\mathcal{T}\log n)$ edges, where $\mathcal{T} = \sum_{e\in E}\widetilde{\boldsymbol{\tau}}_e$.

1. Initiate $\boldsymbol{H}$ as an empty graph.
2. Set sample count $N \leftarrow O(\varepsilon^{-2}\mathcal{T}\log n)$.
3. Repeat N times:
   a. Pick an edge e in G with probability $p_e = \widetilde{\boldsymbol{\tau}}_e/\mathcal{T}$.
   b. Add e to $\boldsymbol{H}$ with new weight $\boldsymbol{w}_e/(Np_e)$.

2.3 Graph Sparsification by Effective Resistances

There are two ways of viewing graph sparsification: either as tossing coins independently on the edges, or sampling a number of them from an overall probability distribution. We take the second view here because it is expensive to access all edges in G^k. The pseudocode of the generic sampling scheme is given in Algorithm 1.

Algorithmically, the sampling step can be implemented by first generating a number uniformly random in $[0, \sum_e \widetilde{\boldsymbol{\tau}}_e]$, (considering we want an edge e to be chosen with probability proportional to a real number $\widetilde{\boldsymbol{\tau}}_e$) and binary searching among the prefix sums of the $\widetilde{\boldsymbol{\tau}}_e$ values until it reaches the edge corresponding to that point. In the RealRAM model [5, 33] of computation, however, this can be done in $O(m)$ preprocessing time and $O(1)$ query time using "pairing" or "aliasing" [6, 24, 39].

The guarantees of this routine require defining effective resistances and leverage scores. Effective resistance is a metric on a graph that is defined by:

$$\mathcal{R}^G_{\text{eff}}(u, v) \stackrel{\text{def}}{=} \boldsymbol{\chi}_{uv}^T \boldsymbol{L}_G^\dagger \boldsymbol{\chi}_{uv}, \tag{3}$$

where $\boldsymbol{L}_G^\dagger$ denotes the pseudoinverse of $\boldsymbol{L}_G$ and $\boldsymbol{\chi}_{uv}$ is the indicator vector with 1 at u and -1 at v. Intuitively, viewing the graph as an electrical network where an edge e acts as a resistor having resistance $1/\boldsymbol{w}_e$, the effective resistance between u and v is the potential difference required between them so that one unit of current flows from u to v.

The effective resistances $\mathcal{R}^G_{\text{eff}}$ are directly related to the statistical leverage scores $\boldsymbol{\tau}$ by the relation $\boldsymbol{\tau}_e = \boldsymbol{w}_e \mathcal{R}^G_{\text{eff}}(e)$. Moreover, these scores are well defined for general matrices, and have a wide range of applications in randomized linear algebra [40, 9, 11]. The guarantees of sampling by weight times effective resistance, or leverage scores, can then be formalized as:

▶ **Lemma 2.** *(Sampling by Upper Bounds on Leverage Scores [37]) Suppose $G = (V, E, w)$ is a graph and $\widetilde{\boldsymbol{\tau}}$ is a vector such that $\widetilde{\boldsymbol{\tau}}_e \geq \boldsymbol{w}_e \mathcal{R}^G_{\text{eff}}(e)$ for every edge e, then, with high probability, any process that simulates the ideal sampling in Algorithm 1 produces an ε-sparsifier of G with $O(\varepsilon^{-2}\mathcal{T}\log n)$ edges in $O(m + \varepsilon^{-2}\mathcal{T}\log^2 n)$ time, where $\mathcal{T} = \sum_{e\in E}\widetilde{\boldsymbol{\tau}}_e$.*

Proof Sketch. A variant (in page 10 of [20]) of the Matrix Chernoff bound [37] states that if $Y = \sum_{i=1}^N Y_i$, $Z = E[Y]$ and $0 \preceq Y_i \preceq RZ$ for every $i \in [k]$ and some scalar R, then for any $\epsilon \in (0, 1)$, it holds

$$Pr\left[(1-\epsilon)Z \preceq Y \preceq (1+\epsilon)Z\right] \geq 1 - 2n \cdot \exp\left\{\frac{-\epsilon^2}{3R}\right\}.$$

Setting Y_i to be the Laplacian of the scaled ith edge added to H in Step 3 of Algorithm 1, we have $Y_i = \dfrac{\boldsymbol{w}_e}{\widetilde{\boldsymbol{\tau}}_e \cdot O(\varepsilon^{-2}\log n)} \chi_e \chi_e^T$ and $Y = \boldsymbol{L}_{\boldsymbol{H}}$. Moreover, $E[\boldsymbol{H}] = G$ and thus $Z = \boldsymbol{L}_G$.

To prove that H is almost always an ϵ-sparsifier of G, it suffices to show that there is a scalar R such that $Y_i \preceq RZ$ for small enough R. Since $\boldsymbol{w}_e \chi_e \chi_e^T \preceq \boldsymbol{\tau}_e \boldsymbol{L}_G$ for every edge e (cf. [9, equation (11) in the proof of Lemma 11]) and by assumption $\boldsymbol{\tau}_e \leq \widetilde{\boldsymbol{\tau}}_e$, it follows that $Y_i \preceq R\boldsymbol{L}_G$ for $R = \Theta(\varepsilon^2/\log n)$. Hence, the desired bound on the failure probability holds.

The runtime follows by noting that in $O(m)$ time we can precompute prefix sums of $\widetilde{\boldsymbol{\tau}}$ and each consecutive edge sample takes $O(\log n)$ time using binary search. ◀

The bound on the number of samples then follows by:

▶ **Fact 3** (Foster's Theorem). *For any undirected graph $G = (V, E, \boldsymbol{w})$, it holds that*

$$\sum_{e \in E} \boldsymbol{w}_e \mathcal{R}_{\text{eff}}^G(u, v) = n - 1.$$

Leverage scores are the preferred objects for defining sampling distributions as they are scale invariant: doubling the weights of all edges does not change their leverages scores. However, we will still make extensive uses of effective resistances because of the need to approximate them across different graphs. Such approximations are difficult to state for leverage scores because spectrally similar graphs may have very different sets of combinatorial edges.

▶ **Fact 4.** *If G and H are graphs such that $\boldsymbol{L}_G \preceq \boldsymbol{L}_H$, then for any vertices u and v we have*

$$\mathcal{R}_{\text{eff}}^H(u, v) \leq \mathcal{R}_{\text{eff}}^G(u, v).$$

Note that this generalizes Rayleigh's monotonicity law, which postulates that the effective resistances can only increase as one removes edges from a graph.

3 Random Walk Sparsification via Walk Sampling

In this section, we describe our improved algorithm for sparsifying random walk polynomials. The main difficulty we need to overcome here is that the actual random walk matrix cannot be constructed explicitly. Instead, we need to simulate the ideal sampling routine shown in Algorithm 1 by constructing nearly tight upper bounds of effective resistances in G^k that can also be efficiently sampled from, without having explicit access to G^k.

To obtain these effective resistances estimates in G^k, the following lemma from [7] provides a helpful starting point.

▶ **Lemma 5.** *[7] For odd k, we have $\frac{1}{2}\boldsymbol{L}_G \preceq \boldsymbol{L}_{G^k} \preceq k\boldsymbol{L}_G$ and for even k, we have $\boldsymbol{L}_{G^2} \preceq \boldsymbol{L}_{G^k} \preceq \frac{k}{2}\boldsymbol{L}_{G^2}$.*

Furthermore, note that Lemma 5 combined with Fact 4 implies for odd k that

$$\mathcal{R}_{\text{eff}}^{G^k}(u, v) \leq 2\mathcal{R}_{\text{eff}}^G(u, v) \tag{4}$$

and for even k that

$$\mathcal{R}_{\text{eff}}^{G^k}(u, v) \leq \mathcal{R}_{\text{eff}}^{G^2}(u, v). \tag{5}$$

Since G^k might be dense, i.e. $E[G^k] = \Theta(n^2)$, it is prohibitive to use (4) and (5) directly. Instead, we upper bound the values with a random walk using the triangle inequality of effective resistances [36, Lemma 9.6.1].

▶ **Fact 6** (Triangle Inequality for Effective Resistances). *For any graph G and any walk $(u_0, u_1, \ldots, u_k)$, we have*

$$\mathcal{R}^G_{\text{eff}}(u_0, u_k) \leq \sum_{0 \leq i < k} \mathcal{R}^G_{\text{eff}}(u_i, u_{i+1}). \tag{6}$$

Now, suppose we have a vector $\widetilde{\boldsymbol{r}}$ that upper bounds the effective resistances, i.e., $\widetilde{\boldsymbol{r}}_e \geq \mathcal{R}^G_{\text{eff}}(e)$ for all e. Then, by Lemma 2 and Fact 6, to sparsify G^k, it suffices to sample a length k random walk in G with probability proportional to

$$\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} \cdot \sum_{0 \leq i < k} \widetilde{\boldsymbol{r}}_{u_i,u_{i+1}}. \tag{7}$$

This distribution has the advantage that it is efficiently computable:

▶ **Lemma 7.** *For any graph $G = (V, E, \boldsymbol{w})$, and any vector $\widetilde{\boldsymbol{r}} \in \mathbb{R}^E$, we can sample length k walks such that the probability of sampling the walk $(u_0, u_1, \ldots, u_k)$ is proportional to*

$$\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} \cdot \sum_{i=0}^{k-1} \widetilde{\boldsymbol{r}}_{u_i,u_{i+1}}$$

using the following procedure:

1. *Pick uniformly at random an index i in the range $[0, k-1]$.*
2. *Choose an edge (u_i, u_{i+1}) with probability proportional to $\boldsymbol{w}_e \widetilde{\boldsymbol{r}}_e$.*
3. *Extend the walk in both directions from u_i and u_{i+1} via two random walks.*

Proof.

(Step 1) Let i be the selected number. The probability of this event is $1/k$.

(Step 2) The probability of selecting an edge (u_i, u_{i+1}) is $\dfrac{\boldsymbol{w}_{u_i,u_{i+1}} \widetilde{\boldsymbol{r}}_{u_i,u_{i+1}}}{\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle}$.

(Step 3) Conditioned on the event that edge (u_i, u_{i+1}) is selected, the probability to sample a walk $(u_0, \ldots, u_k)$ equals

$$\left(\prod_{j=1}^{i} \frac{\boldsymbol{w}_{u_{j-1},u_j}}{\boldsymbol{d}_{u_j}} \right) \cdot \left(\prod_{j=i+1}^{k-1} \frac{\boldsymbol{w}_{u_j,u_{j+1}}}{\boldsymbol{d}_{u_j}} \right) = \frac{\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)}}{\boldsymbol{w}_{u_i,u_{i+1}}}.$$

Thus, summing over all choices of i, and by the total law of probability, the probability of sampling the walk $(u_0, u_1, \ldots, u_k)$ is

$$\sum_{i=0}^{k-1} \frac{1}{k} \cdot \frac{\boldsymbol{w}_{u_i,u_{i+1}} \widetilde{\boldsymbol{r}}_{u_i,u_{i+1}}}{\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle} \cdot \frac{\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)}}{\boldsymbol{w}_{u_i,u_{i+1}}} = \frac{\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)}}{k \langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle} \sum_{i=0}^{k-1} \widetilde{\boldsymbol{r}}_{u_i,u_{i+1}}.$$ ◀

The total number of samples needed by Lemma 2 is given by the summation over all length k random walks, similarly to [7, Lemma 29]. For completeness, we present its proof in Appendix A.

▶ **Lemma 8.** *For any weighted graph $G = (V, E, \boldsymbol{w})$, any $k \in \mathbb{N}_+$, and any vector $\widetilde{\boldsymbol{r}} \in \mathbb{R}^E$, it holds that*

$$\sum_{(u_0,u_1,\ldots,u_k)} \boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} \cdot \sum_{0 \leq i < k} \widetilde{\boldsymbol{r}}_{u_i,u_{i+1}} = k \cdot \langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle. \tag{8}$$

For every odd k, by setting $\widetilde{\boldsymbol{r}}$ to (an approximation of) $\mathcal{R}^G_{\text{eff}}$, yields an efficient sampling procedure due to (8) and Lemma 7.

However, when k is even, Lemma 5 gives a bound in terms of $\mathcal{R}^{G^2}_{\text{eff}}$ (not $\mathcal{R}^G_{\text{eff}}$), i.e. $\mathcal{R}^{G^k}_{\text{eff}}(u,v) \leq \mathcal{R}^{G^2}_{\text{eff}}(u,v)$. Hence, the distribution in Lemma 7 requires an access to the 2-step random walk matrix G^2, which might also be dense and therefore expensive to compute.

Moreover, suppose G is a 2-length path graph $u - v - w$, then $\mathcal{R}^{G^2}_{\text{eff}}(u,v) = +\infty$, since G^2 has only one edge (u,w) (and self-loops). A naive approach to tackle these issues is to substitute $\mathcal{R}^{G^2}_{\text{eff}}$ with $\mathcal{R}^G_{\text{eff}}$. However, this fails shortly since it is not true in general that

$$\mathcal{R}^G_{\text{eff}}(u,v) + \mathcal{R}^G_{\text{eff}}(v,w) \geq \mathcal{R}^{G^2}_{\text{eff}}(u,w). \tag{9}$$

In particular, (9) does not hold for the length 2 path example from above. To verify this, note that $\mathcal{R}^G_{\text{eff}}(u,u) + \mathcal{R}^G_{\text{eff}}(u,v)$ is a finite number, whereas $\mathcal{R}^{G^2}_{\text{eff}}(u,w) = +\infty$ since u and v are disconnected in G^2. For a non-degenerate example, let G be a triangle graph on vertices u, v, w with $\boldsymbol{w}_{uv} = \boldsymbol{w}_{vw} = 1$ and $\boldsymbol{w}_{uw} = 100$. Then, $\mathcal{R}^G_{\text{eff}}(u,u) + \mathcal{R}^G_{\text{eff}}(u,v) \approx 1$ and $\mathcal{R}^{G^2}_{\text{eff}}(u,v) \approx 50$.

We overcome this issue by using effective resistances from the "double cover" of G, instead. The "double cover" $G \times P_2$ is the tensor product of G and a path of length 1. Combinatorially, $G \times P_2$ is a bipartite graph with vertex sets $V^{(A)}, V^{(B)}$ each a copy of V such that for every edge $(u,v) \in G$ we insert in $G \times P_2$ the following two edges: $u^{(A)}v^{(B)}$ and $u^{(B)}v^{(A)}$ with $\boldsymbol{w}_{u^{(A)}v^{(B)}} = \boldsymbol{w}_{u^{(B)}v^{(A)}} = \boldsymbol{w}_{uv}$. The next lemma (proved in Appendix A) relates the effective resistances of G^2 and $G \times P_2$.

▶ **Lemma 9.** *For any vertices u and v in G, it holds*

$$\mathcal{R}^{G^2}_{\text{eff}}(u,v) = \mathcal{R}^{G\times P_2}_{\text{eff}}(u^{(A)}, v^{(A)}),$$

where $u^{(A)}$ and $v^{(A)}$ are the corresponding copies of u and v in $V^{(A)}$, respectively.

Lemma 9 combined with Fact 6, fixes (9) by upper bounding the effective resistance $\mathcal{R}^{G^2}_{\text{eff}}(\cdot)$ with summation of terms $\mathcal{R}^{G\times P_2}_{\text{eff}}(\cdot)$, i.e. for every edge (u,w) in G^2 it holds that

$$\mathcal{R}^{G^2}_{\text{eff}}(u,w) = \mathcal{R}^{G\times P_2}_{\text{eff}}(u^{(A)}, w^{(A)}) \leq \mathcal{R}^{G\times P_2}_{\text{eff}}(u^{(A)}, v^{(B)}) + \mathcal{R}^{G\times P_2}_{\text{eff}}(v^{(B)}, w^{(A)}). \tag{10}$$

Using the preceding results, we design an algorithm with improved sampling count. It takes any procedure that produces effective resistance distribution that dominates the true one (call this an EREstimator), and produces samples that suffice for simulating the ideal sampling algorithm on G^k (cf. Subsection 2.3, Algorithm 1). The pseudocode for this routine is shown in Algorithm 2.

Note that from the perspective of this framework by picking edges with probabilities proportional to $\boldsymbol{w}_e\widetilde{\boldsymbol{r}}_e$, and extending them into walks, the previous result [7] can be viewed as utilizing a simple EREstimator that returns $\widetilde{\boldsymbol{r}}_e = 1/\boldsymbol{w}_e$ as the effective resistance of every edge.

▶ **Theorem 10.** *Given any graph $G = (V, E, \boldsymbol{w})$, any values of k and ε, and any effective resistance estimation algorithm* EREstimator *that produces w.h.p. estimates $\widetilde{\boldsymbol{r}}_e \geq \mathcal{R}^G_{\text{eff}}(e)$ for every edge $e \in E$, then calling* SparsifyG$^{\text{k}}(G, k, \varepsilon,$ EREstimator$)$ *produces an ε-sparsifier of G^k with $O(\varepsilon^{-2}k\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}}\rangle \log n)$ edges in time proportion to the cost of one call to* EREstimator *on a graph of twice the size, plus an overhead of $O(m + \varepsilon^{-2}k^2\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}}\rangle \log^2 n)$.*

Proof. By Lemma 2, it suffices to show that this algorithm simulates the ideal sampling algorithm given in Algorithm 1. Once again we split into the cases of k being odd or even.

Algorithm 2 SparsifyG$^{\text{k}}$ $(G, k, \varepsilon, \text{ERESTIMATOR})$

Input: Graph $G = (V, E, w)$, integer k, error ε, routine ERESTIMATOR that estimates upper bounds for effective resistances of a graph G.
Output: An ε-sparsifier of G^k

1. If k is odd
   a. set $\widetilde{\boldsymbol{r}} \leftarrow \text{ERESTIMATOR}(G)$,
2. else k is even
   a. Set $\widetilde{\boldsymbol{r}}^{(2)} \leftarrow \text{ERESTIMATOR}(G \times P_2)$,
   b. Set $\widetilde{\boldsymbol{r}}_e \leftarrow \widetilde{\boldsymbol{r}}^{(2)}(u^{(A)}, v^{(B)})$, for every edge $e = uv \in E[G]$ (cf. Lemma 9).
3. Set sampling overhead $h \leftarrow O(\varepsilon^{-2} \log n)$ and number of samples $N \leftarrow h \cdot k \cdot \langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle$.
4. Repeat N times
   a. Pick an edge e in G with probability proportional to $\boldsymbol{w}_e \widetilde{\boldsymbol{r}}_e$.
   b. Pick an integer $0 \leq i < k$ uniformly at random, set u_i and u_{i+1} to be endpoints of e.
   c. Perform a random walk by taking i steps from u_i and $k - 1 - i$ steps from u_{i+1}.
   d. Add the edge $(u_0, u_1, \ldots, u_k)$ to H with weight $1/(h \cdot \sum_{0 \leq i < k} \widetilde{\boldsymbol{r}}_{u_i u_{i+1}})$.

When k is odd, Lemma 7 implies that a walk $(u_0, u_1, \ldots, u_k)$ is sampled with probability proportional to $\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} \sum_{0 \leq i < k} \widetilde{\boldsymbol{r}}_{u_i,u_{i+1}}$, where $\widetilde{\boldsymbol{r}}_{u_i,u_{i+1}} \geq \mathcal{R}^G_{\text{eff}}(u_i, u_{i+1})$. Summing over all walks with fixed endpoints (u_0, u_k), by combining (2), (4) and Fact 6, this summation dominates the product $\boldsymbol{w}^{G^k}_{u_0,u_k} \mathcal{R}^{G^k}_{\text{eff}}(u_0, u_k)$. Thus, by Lemma 2 the resulting probability distribution satisfies the statement. The running time and the number of edges in the output sparsifier follow from Lemma 8.

In the case of k being even, by combining Lemma 5 and Lemma 9, we have

$$\mathcal{R}^{G^k}_{\text{eff}}(u, v) \leq \mathcal{R}^{G \times P_2}_{\text{eff}}\left(u^{(A)}, v^{(A)}\right) = \mathcal{R}^{G \times P_2}_{\text{eff}}\left(u^{(B)}, v^{(B)}\right).$$

Also, note that because k is even, each k step walk in G also corresponds to a walk in $G \times P_2$ that starts/ends on the same side, but alternates sides at each step. Using (10) and the symmetry between $u^{(A)}v^{(B)}$ and $u^{(B)}v^{(A)}$, it suffices to sample length k walks with estimated effective resistances satisfying $\widetilde{\boldsymbol{r}}_{uv} \geq \boldsymbol{r}^{G \times P_2}\left(u^{(A)}, v^{(B)}\right)$, for every edge $(u, v) \in G$. The rest of the algorithm follows similarly as in the case of odd k.

To enable picking a neighbor randomly, we need $O(\deg(v))$ preprocessing time for every vertex v, which implies a total preprocessing time of $O(m)$. The extra $O(k \log n)$ in the runtime overhead accounts for performing a random walk of length k, i.e. after preprocessing, a neighboring edge can be sampled using binary search in $O(\log n)$ time. ◀

This reduces the task of sampling edges in G^k to compute good upper bounds for the effective resistances of either the original graph G or of its double cover $G \times P_2$. In the next section we discuss this routine, with focus on density-independent routines.

4 Faster Density Independent Sparsification of Graphs

The monomial sparsification routine from the previous section only requires a distribution that dominates effective resistances for a given graph G. Additionally, we only need a good approximator of G to efficiently compute these approximate effective resistances. The major challenge in keeping the routine density independent is that most numerically oriented

approaches for estimating effective resistances require $O(m \log n)$ time. Instead, a more relevant approach is to utilize "low stretch spanning trees".

Given a graph $G = (V, E, \boldsymbol{w})$, and a tree T, we define the stretch of an edge $e = (u, v) \in E$ w.r.t. T as the ratio of the total resistance on the unique path $\mathcal{P}_T(e)$ between u and v in T to the resistance of e:

$$str_{T,G}(e) \stackrel{\text{def}}{=} \boldsymbol{w}_e \sum_{e' \in \mathcal{P}_T(e)} \frac{1}{\boldsymbol{w}_{e'}}.$$

Extending this definition, the stretch of a subgraph $G'(V', E')$ of G w.r.t. T is given by

$$str_{T,G}(G') \stackrel{\text{def}}{=} \sum_{e \in E'} str_{T,G}(e).$$

We will drop the usage of the second term in the subscript when the underlying graph is obvious from the context.

The advantage of using trees with respect to whom G has low stretch is that the resistance of the path between vertices u and v in the tree can be used as an estimate for the effective resistance of (u, v), and more importantly, the stretch of all edges can be computed using lowest common ancestor queries in only $O(m)$ time [19]. In this context, Lemma 2 can be rewritten as:

▶ **Lemma 11.** *If we have a tree $T \preceq G$, then we can construct an ε-sparsifier of G with $O(\varepsilon^{-2} str_T(G) \log n)$ edges in $O(m)$ time.*

However, we are still left with the issue of constructing such a tree. Abraham and Neiman [2] showed that a tree with stretch $\widehat{O}(m \log n)$ can be constructed in time $\widehat{O}(m \log n)$. This running time does not help our goal of being density-independent. Also, the average stretch is not low enough for the stretches to serve as effective resistance estimates. To tackle both of these issues, we follow the approach used in [26]. We present now a brief overview of this approach and we include the details in Appendix B.

1. Construct a tree T and a graph $\widehat{G}$ obtained by removing $O(m/\log n)$ edges from G such that $str_T(\widehat{G}) \leq \widehat{O}(m \log n)$. This can be computed in $\widehat{O}(m)$ time, using [10, Lemma 5.9] applied with $k = O(\log n)$.
2. Sparsify the removed edges in $O(m)$ time using any standard sparsification method [27, 28] to get H'.
3. To sparsify $\widehat{G}$, construct a series of graphs $\widehat{G}^{(0)}, \widehat{G}^{(1)}, \ldots, \widehat{G}^{(\tau)}$, where $\widehat{G}^{(0)} = \widehat{G}$ and $\widehat{G}^{(\tau)}$ is a graph with low enough stretch such that an $O(1)$-sparsifier $\widehat{H}^{(\tau)}$ of $\widehat{G}^{(\tau)}$ can be constructed in $O(m)$ time.
4. Use the sparsifier $\widehat{H}^{(\tau)}$ to construct an $O(1)$-sparsifier $\widehat{H}^{(\tau-1)}$ of $\widehat{G}^{(\tau-1)}$ and so on, until we get an $O(1)$-sparsifier $\widehat{H}^{(1)}$ of $\widehat{G}^{(1)}$. Every sparsifier $\widehat{H}^{(i)}$ has at most $O(n \log n)$ edges.
5. Repeating Step 4 a final time using effective resistance upper bounds computed from $\widehat{H}^{(0)}$, we compute an ϵ-sparsifier $\widehat{H}$ of $\widehat{G}$. Bringing in the small ϵ only at the last step, allows us to keep the accuracy-related overhead in the intermediate steps at $O(1)$.

This gives us the following results:

▶ **Lemma 12.** *There is a routine that takes a weighted undirected graph G with n vertices, m edges, an error $\epsilon > 0$, and produces in $\widehat{O}(m + \varepsilon^{-2} n \log^4 n)$ time an ϵ-sparsifier of G with $O(\epsilon^{-2} n \log n)$ edges, as well as effective resistance upper bounds $\widetilde{\boldsymbol{r}}$ such that $\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle = \widehat{O}(n \log^2 n)$.*

▶ **Corollary 13.** *There is a combinatorial algorithm that for any graph G on n vertices and m edges, and any error $\epsilon > 0$, produces in $\widehat{O}(m + n\log^6 n)$ time an ϵ-sparsifier of G with $\widehat{O}(\varepsilon^{-2} n \log^2 n)$ edges, as well as effective resistance upper bounds $\widetilde{\boldsymbol{r}}$ such that $\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle = \widehat{O}(n \log^3 n)$.*

The current fastest sparsification routines compute effective resistances via the Johnson-Lindenstrauss transform [35], which in turn requires the use of fast linear system solvers [27].

▶ **Lemma 14.** *Given a graph G, we can compute 2-approximations to its effective resistances in $\widehat{O}(m \log n + n \log^2 n)$ time.*

This runtime bound can be obtained by letting the depth approach n in the proof of Theorem 1.2 on page 49 of [27]. The effective resistances can in turn be extracted from the call to SPARSIFY made at $i = 0$ in the pseudocode in Figure 11 on page 46. We omit details on these steps in the hope that significantly simpler sparsification routines with similar performances will be developed.

Now, we can prove our main result.

Proof of Theorem 1. The upper bound on effective resistances obtained by Lemma 12, when combined with Theorem 10 produces an ϵ-sparsifier of G^k with $\widehat{O}(\epsilon^{-2} kn \log^3 n)$ edges in $\widehat{O}(m + \varepsilon^{-2} k^2 n \log^4 n)$ time. Sparsifying this graph once again using Lemma 14 then leads to the main result as stated in Theorem 1. ◀

The combinatorial guarantees follow similarly from Corollary 13.

Acknowledgements. We thank David Durfee for the various discussions related to applications of these ideas in [14], and the anonymous reviewers of previous versions of this manuscript for their very helpful comments and suggestions.

References

1 Ittai Abraham, David Durfee, Ioannis Koutis, Sebastian Krinninger, and Richard Peng. On fully dynamic graph sparsifiers. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 335–344. IEEE, 2016. Available at: `http://arxiv.org/abs/1604.02094`.

2 Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 395–406. ACM, 2012. Available at: `https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/spanning-full1.pdf`.

3 Sayan Bhattacharya, Monika Henzinger, and Giuseppe F Italiano. Deterministic fully dynamic data structures for vertex cover and matching. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 785–804. SIAM, 2014. Available at: `https://arxiv.org/abs/1412.1318`.

4 Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*, pages 223–234. Springer, 2014.

5 A. Borodin and I. Munro. *The computational complexity of algebraic and numeric problems*. American Elsevier Pub. Co New York, 1975.

6 Karl Bringmann and Konstantinos Panagiotou. Efficient sampling methods for discrete distributions. In Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, editors, *Automata, Languages, and Programming: 39th International Colloquium, ICALP*

2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I, pages 133–144. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. doi:10.1007/978-3-642-31594-7_12.

7 Dehua Cheng, Yu Cheng, Yan Liu, Richard Peng, and Shang-Hua Teng. Efficient sampling for Gaussian graphical models via spectral sparsification. *Proceedings of The 28th Conference on Learning Theory*, pages 364–390, 2015. Available at http://jmlr.org/proceedings/papers/v40/Cheng15.pdf.

8 Yu Cheng and Dehua Cheng. Personal Communication, 2016.

9 Michael B. Cohen, Yin Tat Lee, Cameron Musco, Christopher Musco, Richard Peng, and Aaron Sidford. Uniform sampling for matrix approximation. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science*, ITCS'15, pages 181–190, New York, NY, USA, 2015. ACM. doi:10.1145/2688073.2688113.

10 Michael B. Cohen, Gary L. Miller, Jakub W. Pachocki, Richard Peng, and Shen Chen Xu. Stretching stretch. *arXiv preprint arXiv:1401.2454*, 2014. Available at: https://arxiv.org/abs/1401.2454.

11 Michael B. Cohen, Cameron Musco, and Christopher Musco. Input sparsity time low-rank approximation via ridge leverage score sampling. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1758–1777, 2017. doi:10.1137/1.9781611974782.115.

12 Peter G. Doyle and J. Laurie Snell. *Random Walks and Electric Networks*, volume 22 of *Carus Mathematical Monographs*. Mathematical Association of America, 1984. Available at: https://arxiv.org/abs/math/0001057.

13 David Durfee, Rasmus Kyng, John Peebles, Anup B. Rao, and Sushant Sachdeva. Sampling random spanning trees faster than matrix multiplication. *CoRR*, abs/1611.07451, 2016. Available at: http://arxiv.org/abs/1611.07451.

14 David Durfee, John Peebles, Richard Peng, and Anup B. Rao. Determinant-preserving sparsification of SDDM matrices with applications to counting and sampling spanning trees. *CoRR*, abs/1705.00985, 2017. URL: http://arxiv.org/abs/1705.00985.

15 Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615, July 1987.

16 Andrew V. Goldberg and Robert E. Tarjan. Efficient maximum flow algorithms. *Communications of the ACM*, 57(8):82–89, 2014. Available at: http://cacm.acm.org/magazines/2014/8/177011-efficient-maximum-flow-algorithms/fulltext.

17 Gramoz Goranci, Monika Henzinger, and Mikkel Thorup. Incremental exact min-cut in poly-logarithmic amortized update time. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 46:1–46:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. Full version available at: https://arxiv.org/abs/1611.06500. doi:10.4230/LIPIcs.ESA.2016.46.

18 Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557, 2013. doi:10.1109/FOCS.2013.65.

19 Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *siam Journal on Computing*, 13(2):338–355, 1984.

20 Nick Harvey. Matrix concentration and sparsification. Workshop on "Randomized Numerical Linear Algebra (RandNLA): Theory and Practice", 2012. Available at: http://www.drineas.org/RandNLA/slides/Harvey_RandNLA@FOCS_2012.pdf.

21 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In

Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing, STOC'14, pages 674–683, 2014. Available at: https://arxiv.org/abs/1504.07959.

22 Gorav Jindal and Pavel Kolev. Faster spectral sparsification of laplacian and SDDM matrix polynomials. *CoRR*, abs/1507.07497, 2015. Available at: http://arxiv.org/abs/1507.07497.

23 Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 393–398. ACM, 2012. Available at: https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/sig-alternate.pdf.

24 Donald E. Knuth. *The Art of Computer Programming, Volume 2 (3rd Ed.): Seminumerical Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

25 Ioannis Koutis. Simple parallel and distributed algorithms for spectral graph sparsification. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA'14, pages 61–66, New York, NY, USA, 2014. ACM. Available at: http://arxiv.org/abs/1402.3851. doi:10.1145/2612669.2612676.

26 Ioannis Koutis, Alex Levin, and Richard Peng. Faster spectral sparsification and numerical algorithms for SDD matrices. *ACM Trans. Algorithms*, 12(2):17:1–17:16, December 2015.

27 Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman. Sparsified cholesky and multigrid solvers for connection laplacians. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 842–850. ACM, 2016. Available at http://arxiv.org/abs/1512.01892.

28 Rasmus Kyng, Jakub Pachocki, Richard Peng, and Sushant Sachdeva. A framework for analyzing resparsification algorithms. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA'17, pages 2032–2043, Philadelphia, PA, USA, 2017. Society for Industrial and Applied Mathematics. Available at: https://arxiv.org/abs/1611.06940.

29 Rasmus Kyng and Sushant Sachdeva. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 573–582. IEEE, 2016. Available at: https://arxiv.org/abs/1605.02353.

30 László Lovász. Random walks on graphs: A survey, 1993. Available at: http://www.cs.elte.hu/~lovasz/erdos.pdf.

31 Rasmus Pagh and Charalampos E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters*, 112(7):277–281, 2012.

32 Richard Peng and Daniel A. Spielman. An efficient parallel solver for SDD linear systems. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, STOC'14, pages 333–342, New York, NY, USA, 2014. ACM. Available at http://arxiv.org/abs/1311.3286.

33 Franco P. Preparata and Michael I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

34 Christian Sommer. Shortest-path queries in static networks. *ACM Computing Surveys (CSUR)*, 46(4):45, 2014. Available at: http://www.shortestpaths.com/spq-survey.pdf.

35 D. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011. doi:10.1137/080734029.

36 Daniel A. Spielman. Lecture notes on graphs and networks, October 2007. Available at: http://www.cs.yale.edu/homes/spielman/462/2007/lect9-07.pdf.

37 Joel A. Tropp. User-friendly tail bounds for sums of random matrices. *Found. Comput. Math.*, 12(4):389–434, August 2012. doi:10.1007/s10208-011-9099-z.

38 Charalampos E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 608–617. IEEE, 2008. Available at `http://people.seas.harvard.edu/~babis/tsourICDM08.pdf`.

39 A. J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, April 1974. `doi:10.1049/el:19740097`.

40 David P. Woodruff et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014. Available at: `http://researcher.watson.ibm.com/researcher/files/us-dpwoodru/journal.pdf`.

A Omitted Proofs For Section 3

We give here some additional details on lemmas from Section 3 that are direct consequences of steps in previous works. The total summation of the sampling weights follows from a summation identical to the special case of uniform sampling, as presented in [7, Lemma 29]. More precisely, it is done by evaluating the total weights of all walks with a fixed edge $e \in G$.

Proof of Lemma 8. We first show by induction that the total weights of all length k walks whose i^{th} edge is e is exactly $\boldsymbol{w}_e$.

The base case of $k = 1$ is trivial as only e is a length 1 walk between u_0 and u_1.

The inductive case of $k > 1$ has two cases: $i > 0$ or $i < k - 1$. We consider only the case $i > 0$, as the other one follows by symmetry. Expanding the weight of a length k walk gives:

$$\boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} = \boldsymbol{w}_{(u_0,u_1,\ldots,u_{k-1})} \frac{\boldsymbol{A}_{u_{k-1}u_k}}{\boldsymbol{d}_{u_{k-1}}}.$$

The fact that $i < k - 1$ means that u_k can be any neighbor of u_{k-1}, leading to a sum that cancels the $\boldsymbol{d}_{u_{k-1}}$ term in the denominator. The result then follows from the inductive hypothesis applied to walks of length $k - 1$ that have edge e indexed as the i^{th} walk step:

$$\begin{aligned}\sum_{\substack{(u_0,u_1,\ldots,u_k)\\ e=(u_i,u_{i+1})}} \boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} &= \sum_{\substack{(u_0,u_1,\ldots,u_k)\\ e=(u_i,u_{i+1})}} \boldsymbol{w}_{(u_0,u_1,\ldots,u_{k-1})} \sum_{u_k} \frac{\boldsymbol{A}_{u_{k-1}u_k}}{\boldsymbol{d}_{u_{k-1}}} \\ &= \sum_{\substack{(u_0,u_1,\ldots,u_{k-1})\\ e=(u_i,u_{i+1})}} \boldsymbol{w}_{(u_0,u_1,\ldots,u_{k-1})} \overset{\text{By I.H.}}{=} \boldsymbol{w}_e.\end{aligned}$$

The proof then uses a double counting argument that breaks the summation over edges $e = (u_i, u_{i+1}) \in G$, so as the original summation in (8) becomes equivalent to

$$\sum_{e\in G} \widetilde{\boldsymbol{r}}_e \sum_{0\le i<k} \sum_{\substack{(u_0,u_1,\ldots,u_k)\\ e=(u_i,u_{i+1})}} \boldsymbol{w}_{(u_0,u_1,\ldots,u_k)} = \sum_{e\in G} \widetilde{\boldsymbol{r}}_e \cdot k\boldsymbol{w}_e = k \left\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \right\rangle. \qquad \blacktriangleleft$$

Before we establish an equivalence relation between the effective resistances of the graphs G^2 and $G \times P_2$, we need some notation.

▶ **Definition 15** (Schur Complement)**.** Let $\boldsymbol{M} = \begin{pmatrix} \boldsymbol{M}_{[F,F]} & \boldsymbol{M}_{[F,C]} \\ \boldsymbol{M}_{[C,F]} & \boldsymbol{M}_{[C,C]} \end{pmatrix}$ be a symmetric matrix. The Schur Complement of $\boldsymbol{M}$ induced by removing the block F is defined by

$$Sc(\boldsymbol{M}, F) \overset{\text{def}}{=} \boldsymbol{M}_{[C,C]} - \boldsymbol{M}_{[C,F]} \boldsymbol{M}^{-1}_{[F,F]} \boldsymbol{M}_{[F,C]}.$$

It is known that for any Laplacian $\boldsymbol{M}$ of a graph G, $Sc(M, F)$ is the Laplacian of a graph G^C which is formed by the following iterative process:

- For every vertex $u \in F$
  - For every pair of edges uv_1 and uv_2 in the current graph (with edges from prior steps)
    - Delete edges uv_1 and uv_2, and add a new edge v_1v_2 with weight $\boldsymbol{w}_{uv_1}\boldsymbol{w}_{uv_2}/\boldsymbol{d}_u$, where $\boldsymbol{d}_u$ is the weighted degree of u w.r.t. the current graph.
  - Delete vertex u.

▶ **Lemma 16.** *For every vector* $\boldsymbol{z} = \begin{pmatrix} \boldsymbol{z}_1 \\ 0 \end{pmatrix}$ *it holds that*

$$\boldsymbol{z}_1^T \left(\boldsymbol{D} - \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}\right)^\dagger z_1 = \begin{pmatrix} \boldsymbol{z}_1^T & 0^T \end{pmatrix} \begin{pmatrix} \boldsymbol{D} & -\boldsymbol{A} \\ -\boldsymbol{A} & \boldsymbol{D} \end{pmatrix}^\dagger \begin{pmatrix} \boldsymbol{z}_1 \\ 0 \end{pmatrix}.$$

By symmetry for any vector $\boldsymbol{z} = \begin{pmatrix} 0 \\ \boldsymbol{z}_2 \end{pmatrix}$ *it holds that*

$$\boldsymbol{z}_2^T \left(\boldsymbol{D} - \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}\right)^\dagger \boldsymbol{z}_2 = \begin{pmatrix} 0^T & \boldsymbol{z}_2^T \end{pmatrix} \begin{pmatrix} \boldsymbol{D} & -\boldsymbol{A} \\ -\boldsymbol{A} & \boldsymbol{D} \end{pmatrix}^\dagger \begin{pmatrix} 0 \\ \boldsymbol{z}_2 \end{pmatrix}.$$

In particular, the effective resistances are maintained under Schur complement.

Proof. Consider the linear system

$$\begin{pmatrix} \boldsymbol{D} & -\boldsymbol{A} \\ -\boldsymbol{A} & \boldsymbol{D} \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{pmatrix} \iff \begin{matrix} \boldsymbol{D}\boldsymbol{x} - \boldsymbol{A}y = \boldsymbol{z}_1 \\ -\boldsymbol{A}\boldsymbol{x} + \boldsymbol{D}y = \boldsymbol{z}_2 \end{matrix} \iff \begin{matrix} \boldsymbol{x} = \boldsymbol{D}^{-1}\left(\boldsymbol{z}_1 + \boldsymbol{A}\boldsymbol{y}\right) \\ \boldsymbol{y} = \boldsymbol{D}^{-1}\left(\boldsymbol{z}_2 + \boldsymbol{A}\boldsymbol{x}\right) \end{matrix}.$$

Since $\boldsymbol{z}_2 = 0$, we have

$$\begin{matrix} \boldsymbol{D}\boldsymbol{x} = \boldsymbol{z}_1 + \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{x} \\ \boldsymbol{y} = \boldsymbol{D}^{-1}\boldsymbol{A}\boldsymbol{x} \end{matrix} \implies \boldsymbol{x} = \left(\boldsymbol{D} - \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}\right)^\dagger \boldsymbol{z}_1.$$

and thus

$$\begin{pmatrix} \boldsymbol{z}_1^T & \boldsymbol{z}_2^T \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \boldsymbol{z}_1^T\boldsymbol{x} = \boldsymbol{z}_1^T \left(\boldsymbol{D} - \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}\right)^\dagger \boldsymbol{z}_1.$$ ◀

We can now prove that the effective resistance between u and v in G^2 is the same as the effective resistance between $u^{(A)}$ and $v^{(A)}$ in $G \times P_2$

Proof of Lemma 9. Notice that

$$\boldsymbol{L}_{G^2} = \boldsymbol{D} - \boldsymbol{A}\boldsymbol{D}^{-1}\boldsymbol{A}$$

is the Schur Complement of

$$\boldsymbol{L}_{G\times P_2} = \begin{pmatrix} \boldsymbol{D} & -\boldsymbol{A} \\ -\boldsymbol{A} & \boldsymbol{D} \end{pmatrix}$$

with respect to one half of the vertices, e.g. $V^{(B)}$. The statement follows by Lemma 16. ◀

B Omitted Proofs For Section 4

The following is a detailed exposition of the techniques used to achieve density independent sparsification of a given graph G. The ideas are mainly from [26], but the arguments are tailored to our setting. For the reader's convenience, we present again the scheme overview:

1. Construct a tree T and a graph $\widehat{G}$ obtained by removing $O(m/\log n)$ edges from G such that $str_T(\widehat{G}) \leq \widehat{O}(m \log n)$. This can be computed in $\widehat{O}(m)$ time, using [10, Lemma 5.9] applied with $k = O(\log n)$.
2. Sparsify the removed edges in $O(m)$ time using any standard sparsification method [27, 28] to get H'.
3. To sparsify $\widehat{G}$, construct a series of graphs $\widehat{G}^{(0)}, \widehat{G}^{(1)}, \ldots, \widehat{G}^{(\tau)}$, where $\widehat{G}^{(0)} = \widehat{G}$ and $\widehat{G}^{(\tau)}$ is a graph with low enough stretch such that an $O(1)$-sparsifier $\widehat{H}^{(\tau)}$ of $\widehat{G}^{(\tau)}$ can be constructed in $O(m)$ time.
4. Use the sparsifier $\widehat{H}^{(\tau)}$ to construct an $O(1)$-sparsifier $\widehat{H}^{(\tau-1)}$ of $\widehat{G}^{(\tau-1)}$ and so on, until we get an $O(1)$-sparsifier $\widehat{H}^{(1)}$ of $\widehat{G}^{(1)}$. Every sparsifier $\widehat{H}^{(i)}$ has at most $O(n \log n)$ edges.
5. Repeating Step 4 a final time using effective resistance upper bounds computed from $\widehat{H}^{(0)}$, we compute an ϵ-sparsifier $\widehat{H}$ of $\widehat{G}$. Bringing in the small ϵ only at the last step, allows us to keep the accuracy-related overhead in the intermediate steps at $O(1)$.

B.1 Proof Of Lemma 12

We give now a detailed description of Step 3. The i^{th} graph $\widehat{G}^{(i)}$ in the series is defined by

$$\widehat{G}^{(i)} = \widehat{G} + 2^i \cdot T.$$

We establish next an upper bound on the graph stretch $str_T(\widehat{G}^{(i)})$, for every i. Our proof uses the following notation that highlights the relation between edge stretch and edge weight function.

By definition, the stretch of any tree edge equals 1 and the "on-tree" stretch $str_{T,\widehat{G}^{(i)}}(T)$ has value $n-1$. On the other hand, the stretch of every non-tree edge $e \in \widehat{G}^{(i)} \backslash T$ satisfies

$$str_{T,\widehat{G}^{(i)}}(e) = \boldsymbol{w}_e^{\widehat{G}^{(i)}} \sum_{e' \in \mathcal{P}_T(e)} \left(\boldsymbol{w}_{e'}^{\widehat{G}^{(i)}}\right)^{-1} = \boldsymbol{w}_e^{\widehat{G}} \sum_{e' \in \mathcal{P}_T(e)} \left((2^i+1)\boldsymbol{w}_{e'}^{\widehat{G}}\right)^{-1} \leq 2^{-i} \cdot str_{T,\widehat{G}}(e).$$

Moreover, since

$$str_{T,\widehat{G}^{(i)}}(\widehat{G}\backslash T) \leq 2^{-i} \cdot str_{T,\widehat{G}}(\widehat{G}\backslash T) \leq 2^{-i} \cdot str_{T,\widehat{G}}(\widehat{G}) = O(2^{-i} \cdot m \log n),$$

it follows that the total stretch of graph $\widehat{G}^{(i)}$ w.r.t. T is bounded by

$$str_T(\widehat{G}^{(i)}) = str_{T,\widehat{G}^{(i)}}(\widehat{G}\backslash T) + str_{T,\widehat{G}^{(i)}}(T) \leq O(2^{-i} \cdot m \log n).$$

Therefore, the initial graph $\widehat{G}^{(\tau)}$ for $\tau = \Omega(\log \log n)$ has total stretch

$$str_T(\widehat{G}^{(\tau)}) = \widehat{O}(m/\log^2 n).$$

Using Lemma 11, we can compute in $O(m)$ time an $O(1)$-sparsifier $G'^{(\tau)}$ of $\widehat{G}^{(\tau)}$ with $\widehat{O}(m/\log n)$ edges. Invoking any standard nearly-linear time sparsification algorithm on $G'^{(\tau)}$ then gives us in $O(m)$ time a $O(1)$-sparsifier $\widehat{H}^{(\tau)}$ of $G^{(\tau)}$ with $O(n \log n)$ edges.

We present now the TreeSparsify routine which is used in Step 4 and Step 5.

Algorithm 3 TreeSparsify$(G, G', \kappa, \varepsilon)$

Input: Graph $G = (V, E, \boldsymbol{w})$ with κ-sparsifier G', and error $\varepsilon > 0$.
Output: $\widetilde{G}$ that is an ε-sparsifier of G.

1. Compute a low stretch spanning tree T of G'.
2. Compute an upper bound on all leverage scores $\widetilde{\boldsymbol{\tau}}$ of G using [19].
3. Sample $O(\varepsilon^{-2} str_T(G) \log n)$ edges of G using IdealSample$(G, \varepsilon, \widetilde{\boldsymbol{\tau}})$ (cf. Algorithm 1).

▶ **Lemma 17.** *Given a κ-sparsifier G' of G and $\varepsilon > 0$,* TreeSparsify$(G, G', \kappa, \varepsilon)$ *produces an ε-sparsifier of G with $\widehat{O}(\varepsilon^{-2}\kappa\,|E(G')|\log^2 n)$ edges in $\widehat{O}(m + \varepsilon^{-2}\kappa\,|E(G')|\log^3 n)$ time.*

Proof. To apply Lemma 2, we have to compute a vector $\widetilde{\boldsymbol{r}} \geq \mathcal{R}^G_{\text{eff}}$ and give an upper bound on $\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle$. Since $\boldsymbol{L}_G \preceq \boldsymbol{L}_{G'}$, by [26, Lemma 6.4] we have $str_T(G) \leq str_T(G')$. Additionally, since $\boldsymbol{L}_T \preceq \boldsymbol{L}_{G'} \preceq \kappa \boldsymbol{L}_G$, it follows that

$$\widetilde{\boldsymbol{r}} \stackrel{\text{def}}{=} \kappa \cdot \mathcal{R}^T_{\text{eff}} \geq \mathcal{R}^G_{\text{eff}}. \tag{11}$$

Using the above statements, and the low stretch spanning tree construction of Abraham and Neiman [2], we obtain

$$\langle \boldsymbol{w}, \widetilde{\boldsymbol{r}} \rangle = \kappa \cdot str_T(G) \leq \kappa \cdot str_T(G') = \widehat{O}(\kappa\,|E(G')|\log n).$$

The statement follows by Lemma 2. ◀

We present now the core iterative procedure underlying Step 4 and Step 5:

(i) Let $\delta > 0$ be an error parameter. In Step 4, we set $\delta = O(1)$, whereas $\delta = \epsilon$ in Step 5.
(ii) Straightforward checking shows that by construction $\widehat{H}^{(i+1)}$ is a $O(1)$-sparsifier of $\widehat{H}^{(i)}$.
(iii) Compute a $\delta/2$-sparsifier $G'^{(i)}$ of $\widehat{G}^{(i)}$ with $\widehat{O}(\delta^{-2} n \log^3 n)$ edges in $\widehat{O}(m + \delta^{-2} n \log^4 n)$ time, calling TreeSparsify$(\widehat{G}^{(i)}, \widehat{H}^{(i+1)}, O(1), \delta)$. The guarantees follow by Lemma 17.
(iv) Compute a $\delta/2$-sparsifier $\widehat{H}^{(i)}$ of $G'^{(i)}$ with $O(\delta^{-2} n \log n)$ edges in $\widehat{O}(\delta^{-2} n \log^4 n)$ time, using Lemma 14 and Lemma 2. Thus, $\widehat{H}^{(i)}$ is a δ-sparsifier of $\widehat{G}^{(i)}$.

We analyze now the runtime of Steps 4 and 5. In Step 4, there are $O(\log\log n)$ calls to TreeSparsify each with $\delta = O(1)$. Thus, by Lemma 17, Step 4 runs in $\widehat{O}(m + n\log^4 n)$ time. In Step 5, we set $\delta = \epsilon$. Then, by Lemma 14 and Lemma 2, the runtime of Step 5 is bounded by $\widehat{O}(m + \epsilon^{-2} n \log^4 n)$.

B.2 Proof Of Corollary 13

We use purely combinatorial constructions of graph sparsifiers that are based on spanners [23, 25, 28]. We summarize these results in the following lemma.

▶ **Lemma 18** ([28, Theorem 4.1])**.** *Given G and error $\varepsilon > 0$, we can compute an ε-spectral sparsifier of G with $\widehat{O}(n\log^2 n)$ edges in $\widehat{O}(m\log^2 n)$ time.*

We show now that the algorithm in Lemma 18 applied to our sparsification scheme yields Corollary 13. We argue in a similar manner as in the routine calling numerical sparsifiers, outlined in Corollary 12. Here, in contrast, every sparsifier $\widehat{H}^{(i+1)}$ has $\widehat{O}(n\log^2 n)$ edges, and thus every sparsifier $G'^{(i)}$ has $\widehat{O}(n\log^4 n)$ edges. Hence, every consecutive re-sparsification call yield a sparsifier $\widehat{H}^{(i)}$ with $\widehat{O}(n\log^2 n)$ edges in $\widehat{O}(n\log^6 n)$ time.